mchalama_assignment2

Manoj Kumar Chalamala

3/3/2020

Load Keras Library

```
library(keras)
```

Download the IMDB data as raw text

let’s collect the individual training reviews into a list of strings, one string per review. You’ll also collect the review labels (positive / negative) into a labels list

```
imdb_dir <- "C:/Users/ManojKumar Chalamala/Downloads/aclImdb.tar/aclImdb"
train_dir <- file.path(imdb_dir, "train")
labels <- c()
texts <- c()
for (label_type in c("neg", "pos")) {
  label <- switch(label_type, neg = 0, pos = 1)
  dir_name <- file.path(train_dir, label_type)
  for (fname in list.files(dir_name, pattern = glob2rx("*.txt"),
                           full.names = TRUE)) {
    texts <- c(texts, readChar(fname, file.info(fname)$size))
    labels <- c(labels, label)
  }
}
```

Tokenize the data

By cutting reviews after 150 words, By Restricting Samples to 100 By Validating on 10000 samples By COnsidering only the top 10000 words

```
maxlen <- 150                 # We will cut reviews after 150 words
training_samples <- 100       # We will be training on 100 samples
validation_samples <- 10000   # We will be validating on 10000 samples
max_words <- 10000            # We will only consider the top 10,000 words in the data
set

tokenizer <- text_tokenizer(num_words = max_words) %>%
  fit_text_tokenizer(texts)

sequences <- texts_to_sequences(tokenizer, texts)
word_index = tokenizer$word_index

cat("Found", length(word_index), "unique tokens.\n")
```

```
## Found 88584 unique tokens.
```

```
data <- pad_sequences(sequences, maxlen = maxlen)
y_data <- as.array(labels)
labels <- as.array(labels)
cat("Shape of data tensor:", dim(data), "\n")
```

```
## Shape of data tensor: 25000 150
```

```
cat('Shape of label tensor:', dim(labels), "\n")
```

```
## Shape of label tensor: 25000
```

```
# Split the data into a training set and a validation set
# But first, shuffle the data, since we started from data
# where sample are ordered (all negative first, then all positive).
set.seed(123)
indices <- sample(1:nrow(data))
training_indices <- indices[1:training_samples]
validation_indices <- indices[(training_samples + 1):
                              (training_samples + validation_samples)]

x_train <- data[training_indices,]
y_train <- labels[training_indices]

x_val <- data[validation_indices,]
y_val <- labels[validation_indices]
```

Preprocess the embeddings

```
glove_dir = 'C:/Users/ManojKumar Chalamala/Downloads/glove.6B'
lines <- readLines(file.path(glove_dir, "glove.6B.100d.txt"))
embeddings_index <- new.env(hash = TRUE, parent = emptyenv())
for (i in 1:length(lines)) {
  line <- lines[[i]]
  values <- strsplit(line, " ")[[1]]
  word <- values[[1]]
  embeddings_index[[word]] <- as.double(values[-1])
}

cat("Found", length(embeddings_index), "word vectors.\n")
```

```
## Found 400000 word vectors.
```

Build an embedding matrix to load into an embedding layer

```
embedding_dim <- 100
embedding_matrix <- array(0, c(max_words, embedding_dim))

for (word in names(word_index)) {
  index <- word_index[[word]]
  if (index < max_words) {
    embedding_vector <- embeddings_index[[word]]
    if (!is.null(embedding_vector))
      # Words not found in the embedding index will be all zeros.
      embedding_matrix[index+1,] <- embedding_vector
  }
}
```

Define a Model

```
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_words, output_dim = embedding_dim,
                  input_length = maxlen) %>%
  layer_flatten() %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

summary(model)
```

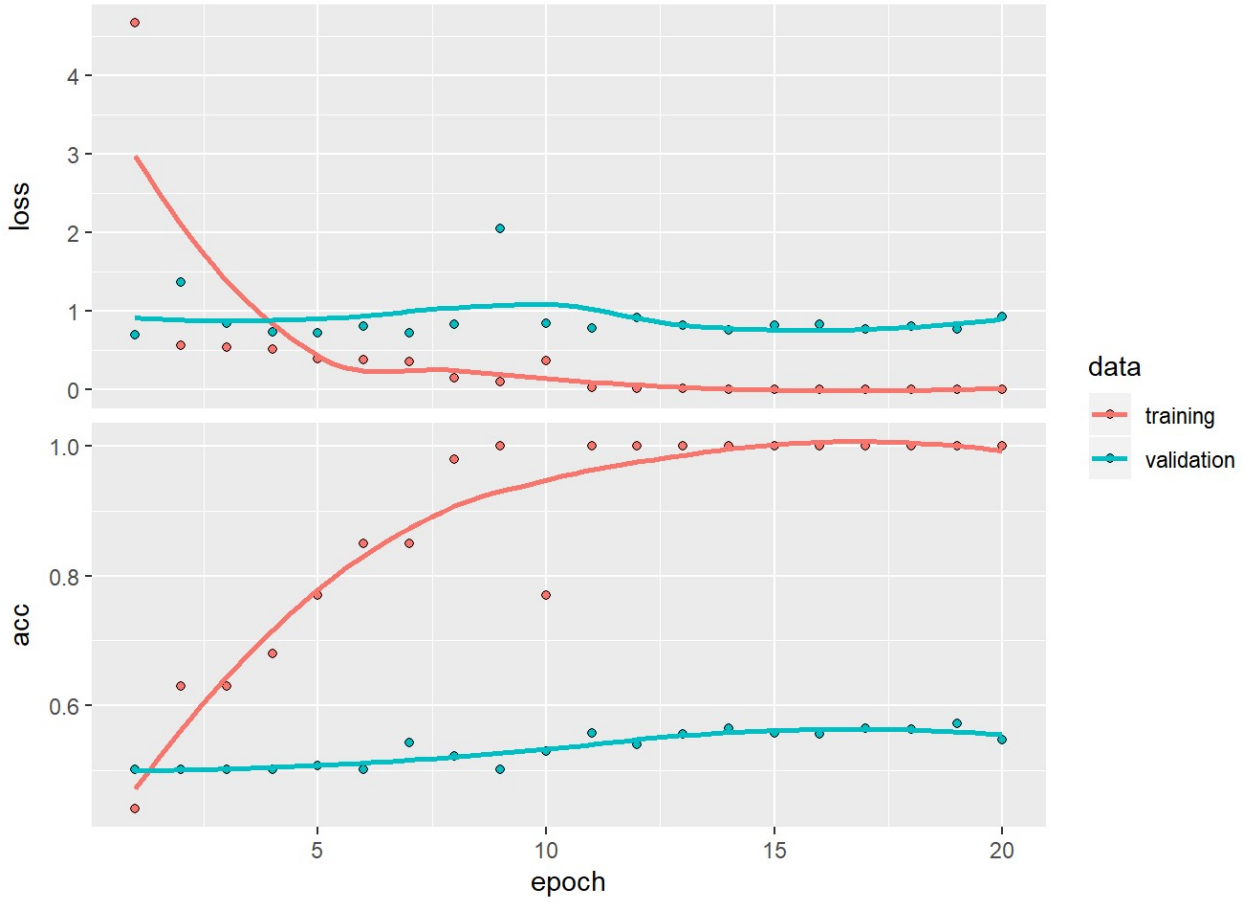
```
## Model: "sequential"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## embedding (Embedding)               (None, 150, 100)                1000000
## ________________________________________________________________________________
## flatten (Flatten)                   (None, 15000)                   0
## ________________________________________________________________________________
## dense (Dense)                       (None, 32)                      480032
## ________________________________________________________________________________
## dense_1 (Dense)                     (None, 1)                       33
## ================================================================================
## Total params: 1,480,065
## Trainable params: 1,480,065
## Non-trainable params: 0
## ________________________________________________________________________________
```

Load the pretrained GloVe embeddings in the model

```
get_layer(model, index = 1) %>%
  set_weights(list(embedding_matrix)) %>%
  freeze_weights()

model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)

#evaluating on train model
history <- model %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_data = list(x_val, y_val)
)
plot(history)
```



Evaluating on test data

```
test_dir <- file.path(imdb_dir, "test")
labels <- c()
texts <- c()
for (label_type in c("neg", "pos")) {
  label <- switch(label_type, neg = 0, pos = 1)
  dir_name <- file.path(test_dir, label_type)
  for (fname in list.files(dir_name, pattern = glob2rx("*.txt"),
                           full.names = TRUE)) {
    texts <- c(texts, readChar(fname, file.info(fname)$size))
    labels <- c(labels, label)
  }
}
sequences <- texts_to_sequences(tokenizer, texts)
x_test <- pad_sequences(sequences, maxlen = maxlen)
y_test <- as.array(labels)
```

```
model %>% fit(
  x_train,
  y_train,
  epochs = 2,
  batch_size = 32)
result <- model %>%  evaluate(x_test, y_test)
result # Test Accuarcy is 56%
```

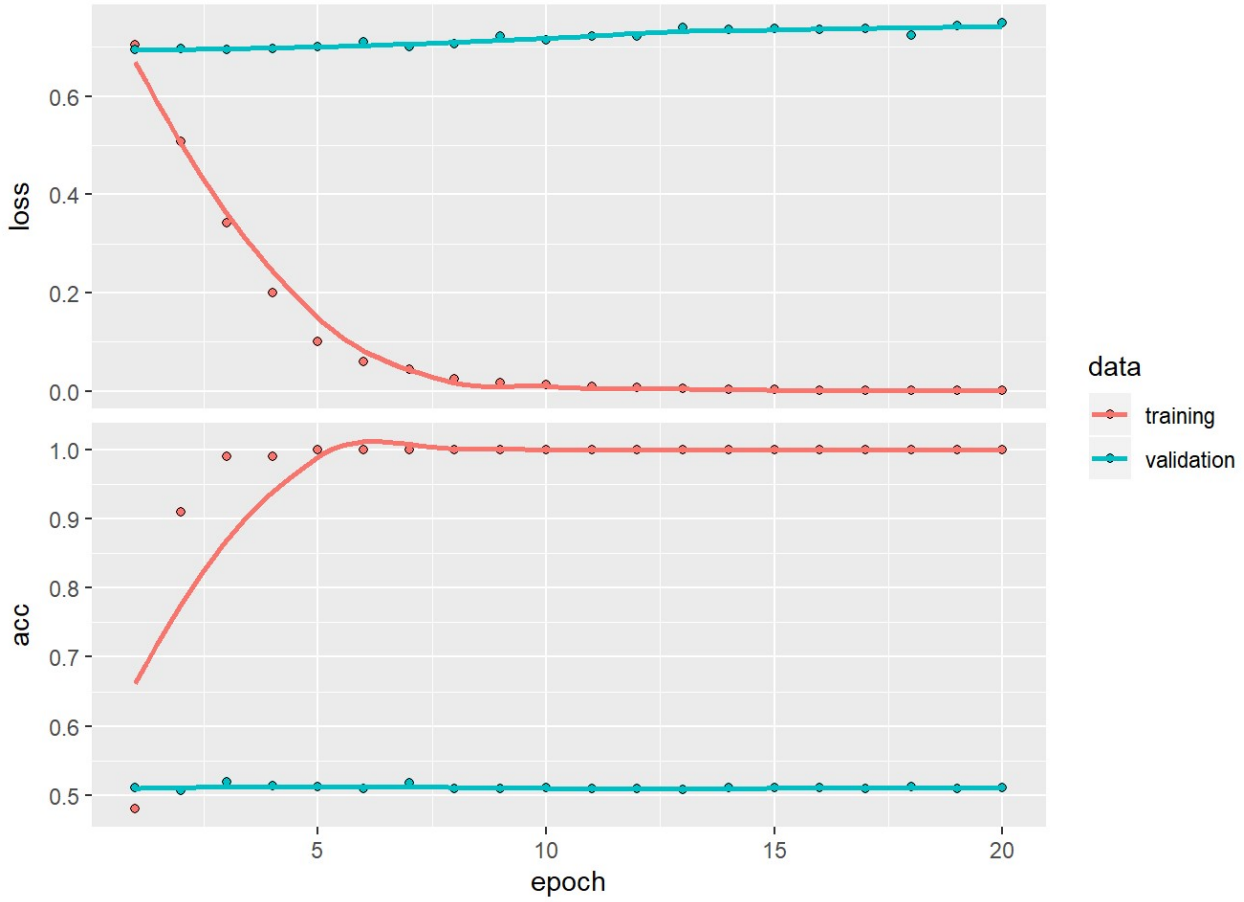
```
## $loss
## [1] 0.7949171
##
## $acc
## [1] 0.56948
```

Define a model by considering an embedding layer and classifier on the IMDB data

```
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_words, output_dim = embedding_dim,
                  input_length = maxlen) %>%
  layer_flatten() %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)


history1 <- model %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_data = list(x_val, y_val)
)
plot(history1)
```



Evaluating on Test Data

```
model %>% fit(
  x_train,
  y_train,
  epochs = 2,
  batch_size = 32)
result1 <- model %>%  evaluate(x_test, y_test)
result1 # Test Accuracy is 50%
```

```
## $loss
## [1] 0.7678634
##
## $acc
## [1] 0.5072
```

Therefore the Model Accuracy of Pretrained embedding layer is better than the model accuracy of considering an embedding layer and classifer on IMDB data. Performance also depends on the number of training samples we choose. It gives a slightly better performance with fewer training samples.